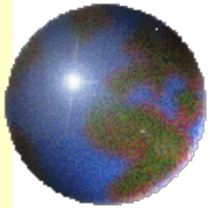
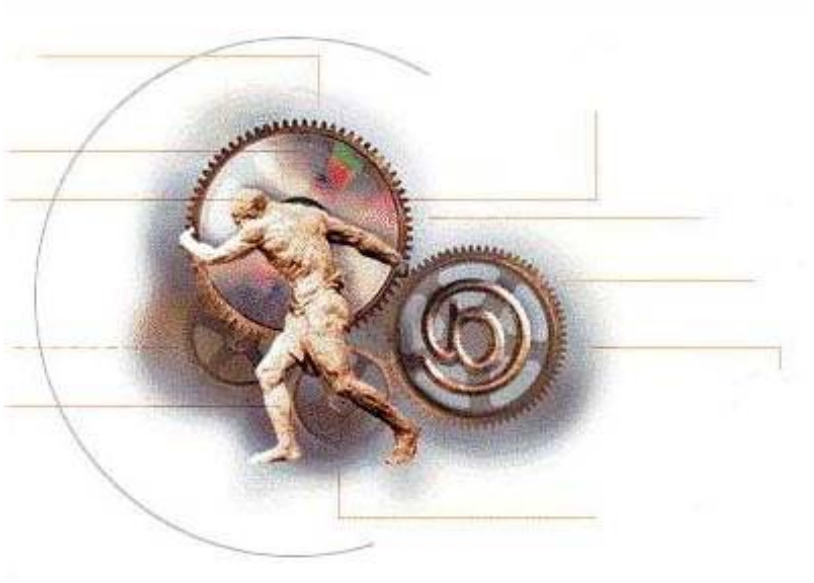


資料結構(Data Structures)

Course 4: Link Lists (鏈結 串 列)



授課教師:陳士杰

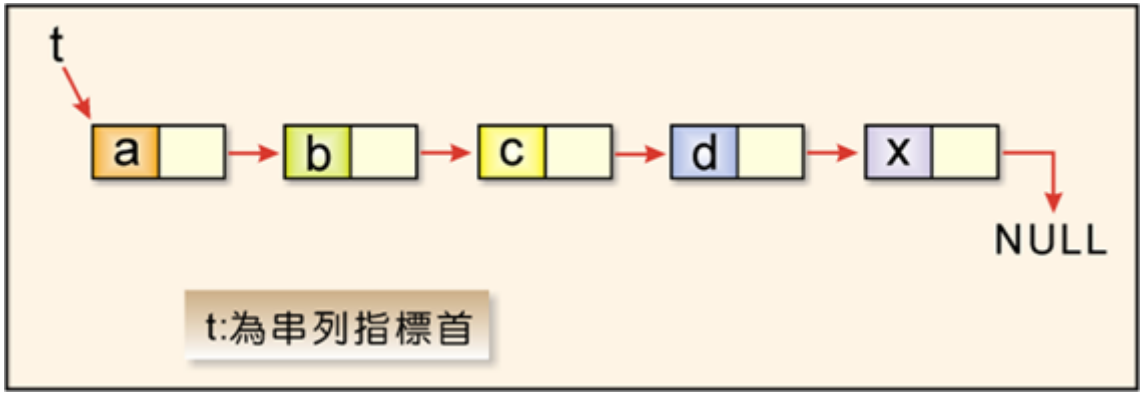
國立聯合大學　資訊管理學系

Outlines

- 本章重點
  - Link List's Def.
  - 與 Array 的比較
  - Link List之基本操作 (Insert, Delete)
  - Link list的種類:
    - Single Link List (單向鏈結串列)
    - Circular Link List (環狀鏈結串列)
    - Double Link List (雙向鏈結串列)
  - Link List常見的三個運作
    - Length (計算串列長度)
    - Concatenate (連結兩個串列)
    - Invert (反轉)
  - Storage Pool

Linked List Concepts

- Def: 由一組**節點 (Node)**所組成的**有序串列**，各Node除了Data欄之外，另外有 ≥ 1 個Link欄 (或稱 Pointer)，用以指向其它Node之位址。
  - 例：單向鏈結串列之節點：



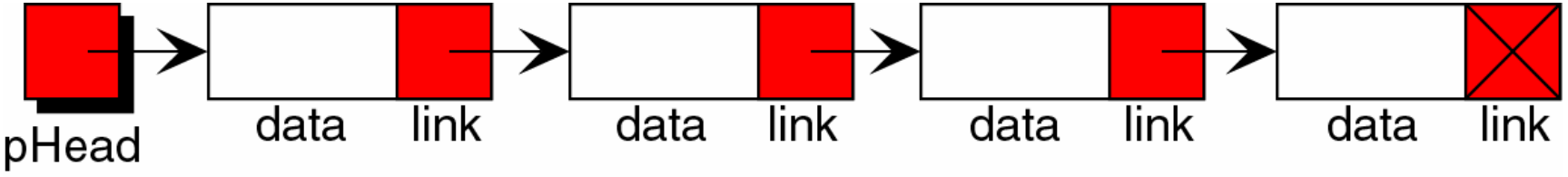

```
struct node                 /* 串列結構宣告 */
{
   int data;                /* 資料 */
   struct node *next;       /* 指標，指向下一個節點 */
};
```

- Linked list的特質:
  - 各Node不一定要佔用連續的Memory空間
  - 各Node之型態 (Data Type) 不一定要相同
  - 僅支援Sequential Access
  - Insert/Delete Node 容易

- Link list的種類:
  - Single Link List (單向鏈結串列)
  - Circular Link List (環狀鏈結串列)
  - Double Link List (雙向鏈結串列)

Single Linked List (單向鏈結串列)

- Def: 由一組**節點 (Node)**所組成的**有序串列**，各Node除了Data欄之外，另外有 1 個Link欄 (或稱 Pointer)，用以指向下一個Node之位址。
  - 範例：名為 "pHead" 之單向鏈結串列 (視首節點或首指標之名稱為何而定)



(a) A linked list with a head pointer: pHead

(b) An empty linked list

Linked List Algorithms

- Create List
- Insert Node
- Delete Node
- 在一些鏈節串列運作中，我們可能會設定其它輔助用的指標!!

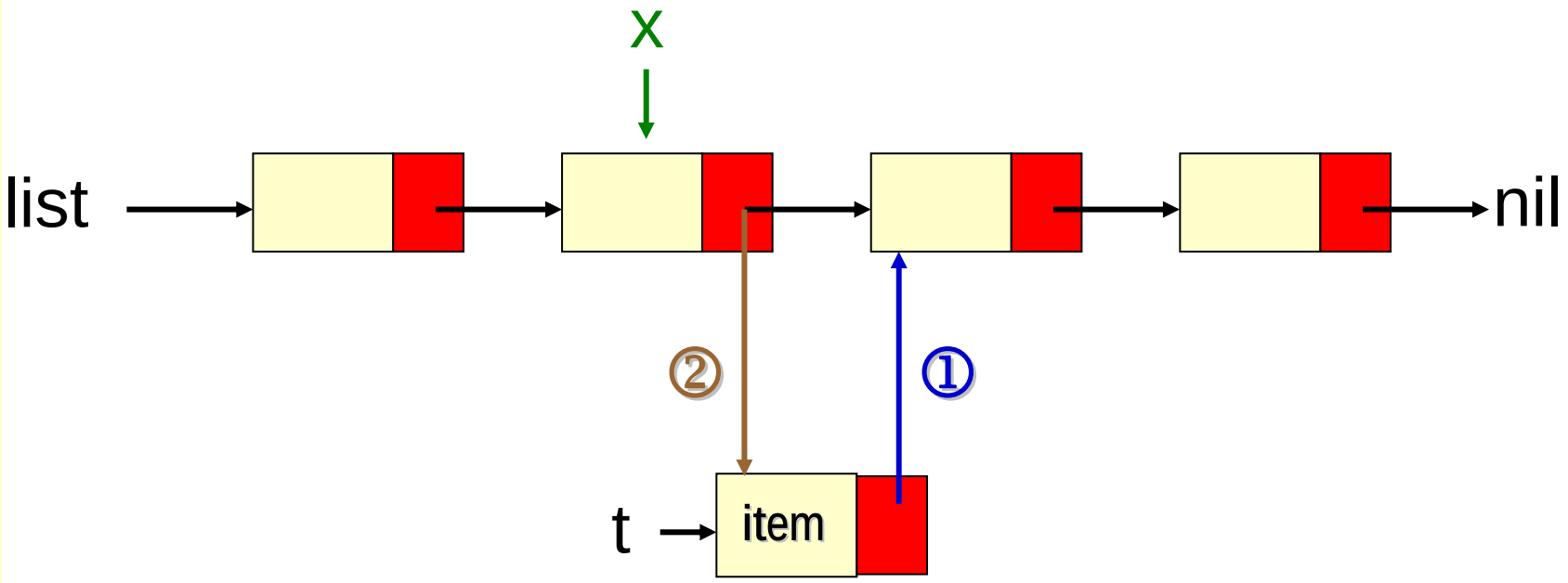
Create List

- 以 “list” 為名的空串列:
  - 以指標作為串列首 :

list ⟶ null

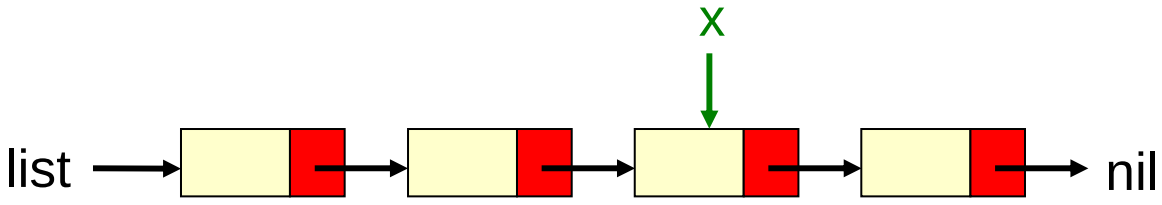
Insert Node

- 將資料item插入在串列list中的節點 x 之後。



```
begin
    New(t); //跟系統要求記憶體空間以配置一個新節點，且讓指標 t 指向該節點
    t→data = item; //把item塞到這個新節點t的Data欄位中
① t→link = x→link; //先把新節點t掛到x節點之後!!
② x→link = t; //再將x節點接到節點t
end
```

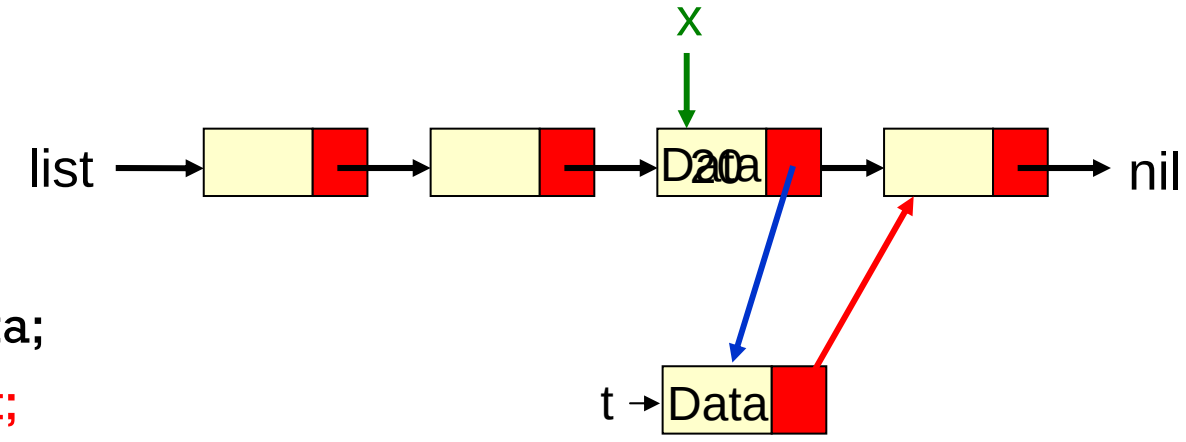
- 例:插入資料20在節點 x 之前



限制:不准用迴圈指令，且動作需於O(1)常數時間完成。

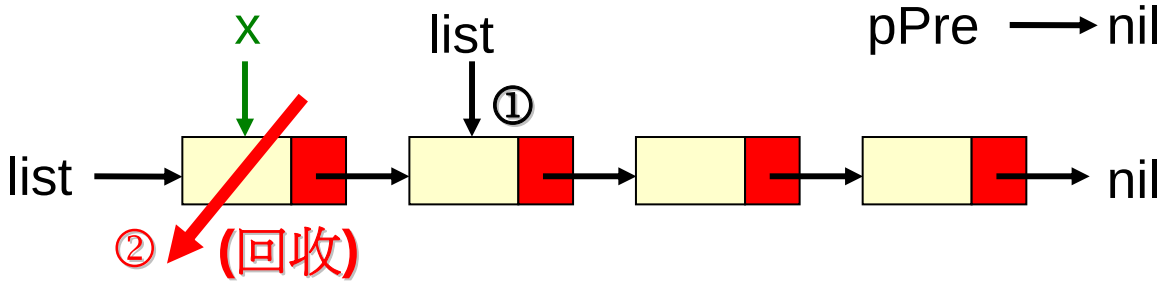
Ans:

```
begin
   New(t);
   t→data = x →data;
   t→link = x →link;
   x →link = t;
   x →data = 20;
   x = t;
end;
```

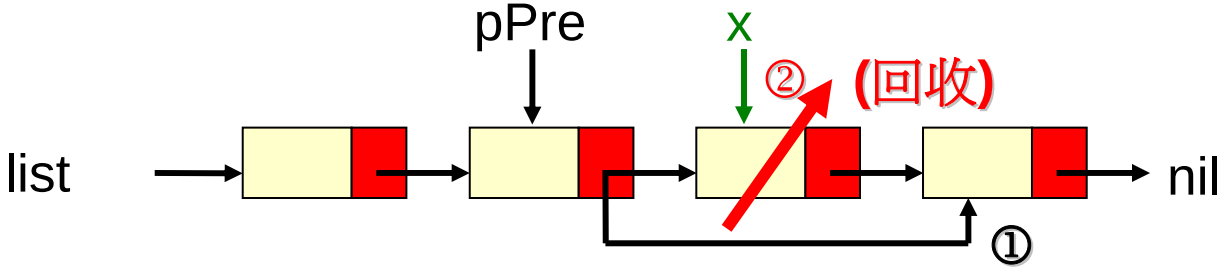


Delete Node

- 刪除鏈節串列的節點時，會先 ① 改變串列指標，以做到**邏輯性移除**(Logically Remove)的動作， ② 再將要被移除的節點做**實際刪除**(Physically Delete) 的動作，讓節點從記憶體中刪除。
- 需要兩個輔助指標 x 與 pPre。x用以指向**欲刪除的節點**，pPre指向**節點x所在之前一個node**。
- 例: 刪除串列 list 中之節點 **x** ，
  - Case 1: (要砍**第一個node**)--若x為第一個node，則令pPre = nil。



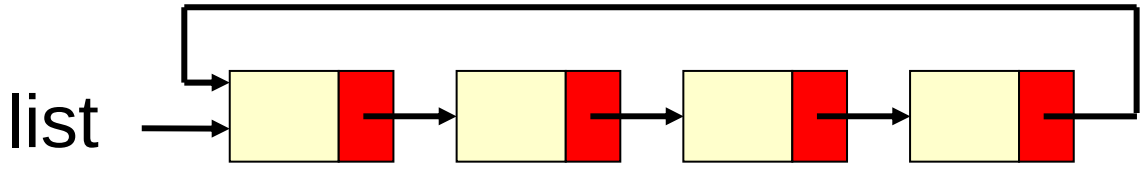
  - Case 2: (要砍**串列中間的node**)



```
begin
 ① if (pPre = nil) then //表示 x 為第一個節點
       list = x→link;
    else //表示 x 節點是中間節點
       pPre→link = x→link;
 ② Ret(x); //將 "節點 x" 回收. 有的書是用delete(x), release(x), dispose(x)…etc.
end
```

Circularly Linked Lists (環狀鏈結串列)

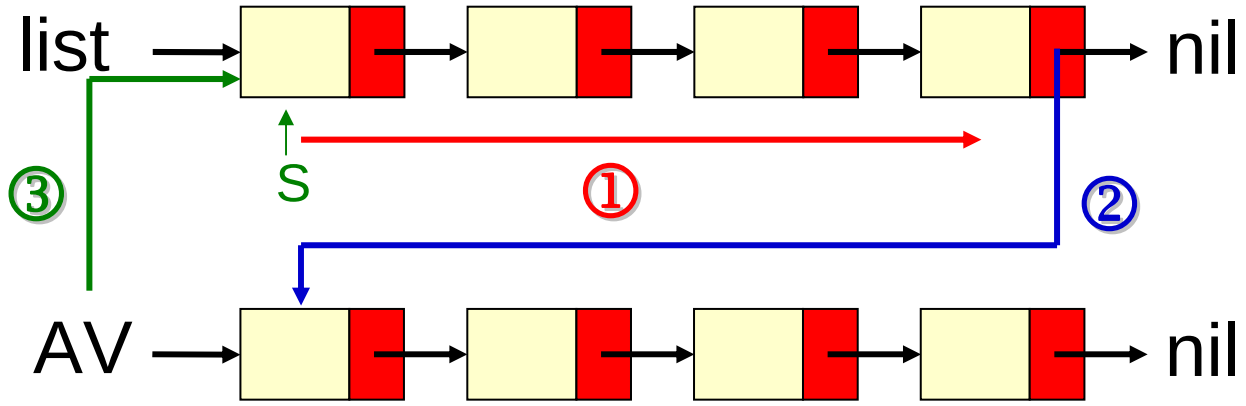
- Def: 將Single link list中，最後一個node的指標指回第一個node.
- 圖示:



- 特質:
  - 不論從哪個Node開始，必定能夠拜訪 (經過) 所有Node.
    - 並無明顯的頭尾節點
    - Single Link List必須從頭開始才可以
  - 回收整個串列的時間為**O(1)**. (即: 與Link list長度無關)
    - Single Link List回收時間為O(n). 其中, n為長度或Node數目.

回收時間的比較(單向 vs. 環狀)

- Single Linked List:



Procedure Release list(list: pointer to S.L.)

begin

if (list≠nil) then

S = list;

① **while (S→link ≠ nil) do** **//while迴圈會讓S一直前進至最後一節點**

S = S→link;

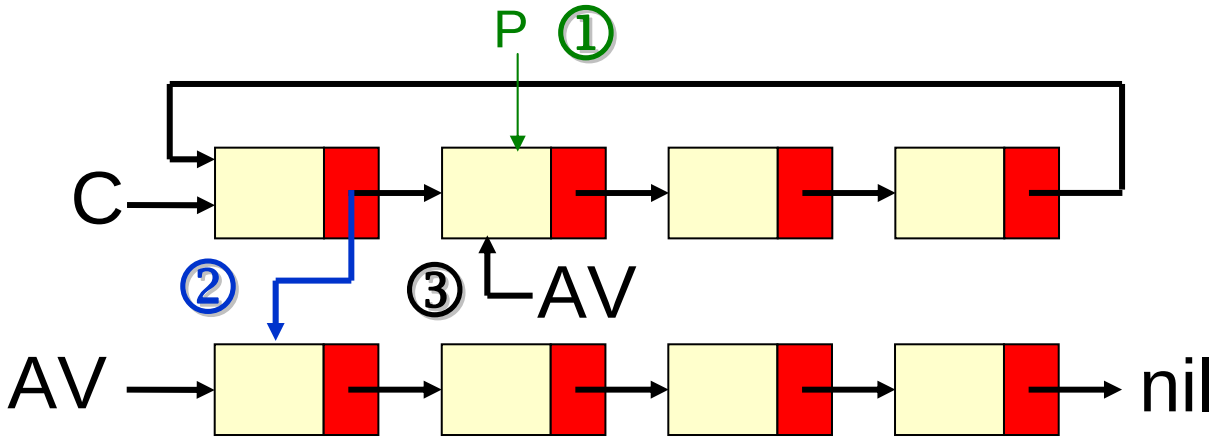
② **S→link = AV;**

③ **AV = list; //回收節點至AV list 中**

end

∴ Time = O(n)

Circular Linked List C:



Procedure Release C(C: pointer to S.L.)

begin

if (C≠nil) then

① **p = C→link;**

② **C→link = AV;**

③ **AV = P;**

end

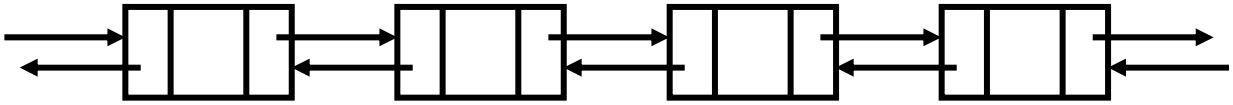
//只動了三個指標!!

∴ Time = O(1)

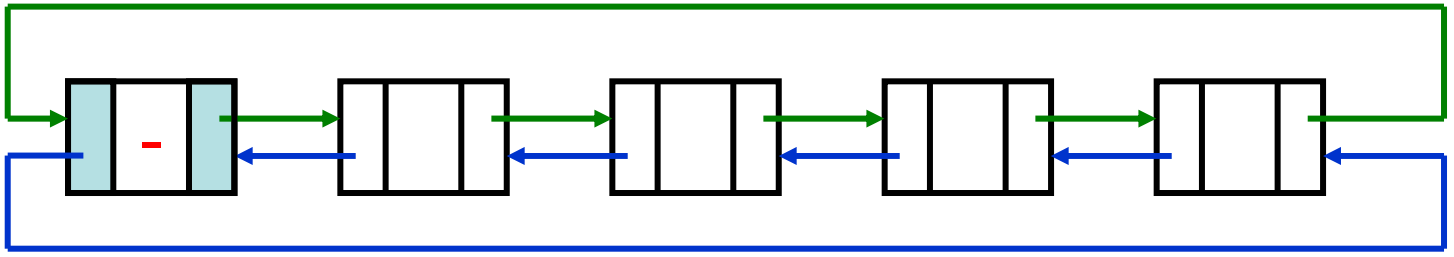
Doubly Linked List (雙向鏈結串列)

- Def: 串列中各節點除了Data欄之外，另外有 2 個Link欄，用以指向前一個與下一個節點之位址。
- 圖示:

  - **LLink**: Pointer指向前一個Node.
  - **RLink**: Pointer指向後一個Node.



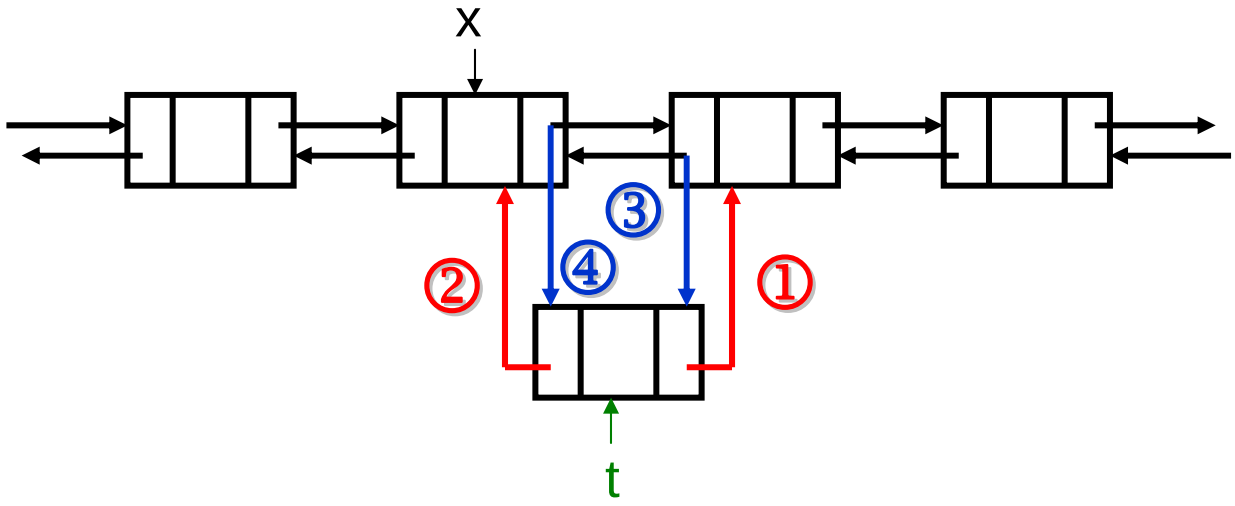
- 一般使用雙向鏈結串列時，通常會加入**串列首 (Head Node)**，此Node不存資料。



- 此時，雙向鏈結串列可視為2個Circular Link List.

Doubly Linked List Insertion

- 插入Node t 在Node x之後



begin

① t→RLink = x→RLink;

② t→LLink = x;

③ (x→RLink)→LLink = t;

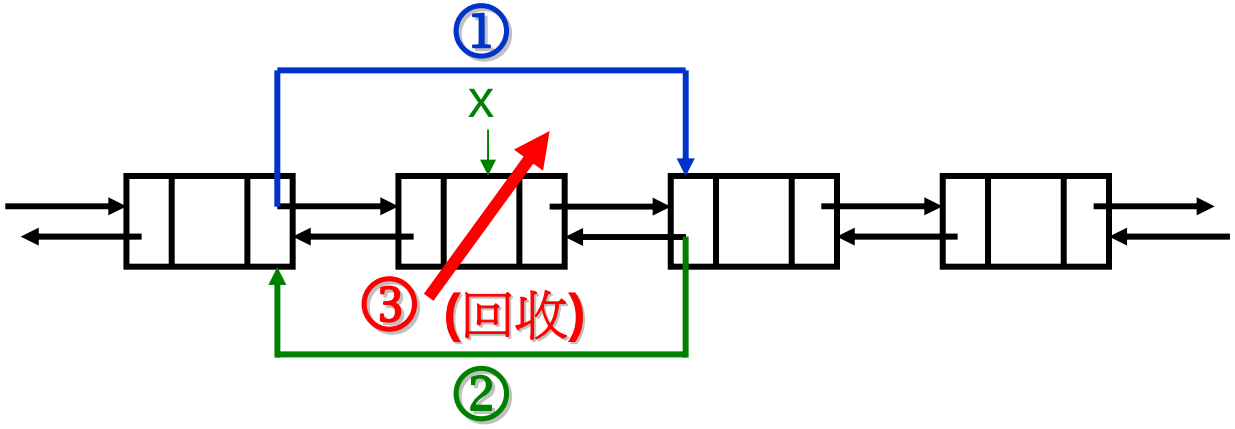
④ x→RLink = t;

※須改變4個指標

end

Doubly Linked List Deletion

- 刪除一個節點 x



begin

① (x→LLink)→RLink = x→RLink;

② (x→RLink)→LLink = x→LLink ;

③ Ret(x);

※須改變2個指標

end

Double linked list 與 Single linked list 的比較

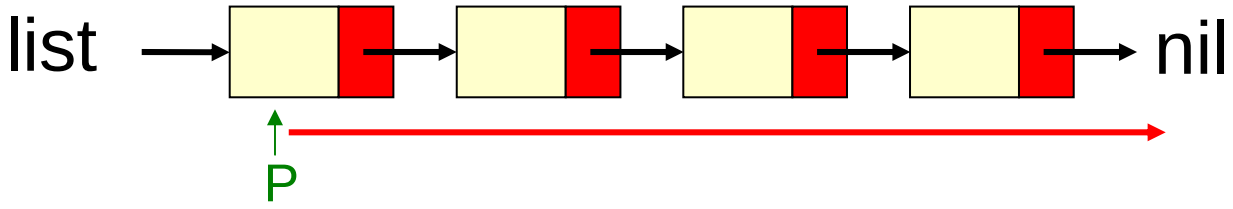
	Double	Single	
優			缺
缺			優

Link list三個常見的動作

- 動作:
  - Length (求串列長度)
  - Concatenate (串列連結)
  - Invert (反轉)
- 討論對象:
  - Single linked list
  - Circular linked list
- 共有6個主要的Algorithm

Length

- For "Single linked list":

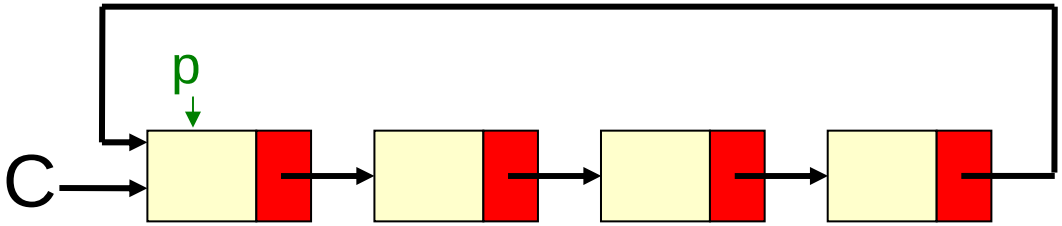


count++; //p不爲空 (nil) 時，則count加1，以讓p往下跑

```
Procedure Length list(list: pointer to S.L.)
begin
    count = 0;
    p = list;
    while (p ≠ nil) do
    begin
        count++;
        p = p→link;
    end;
    return count;
end
```

∴ Time: O(n)

- For "Circular linked list C":

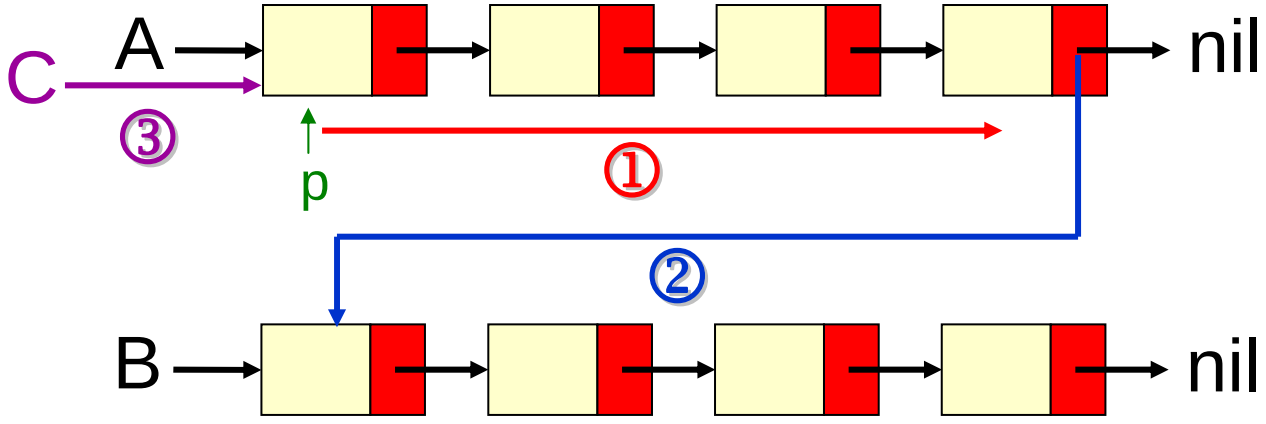



```
Procedure Length C(C: pointer to S.L.)
begin
    if (C = nil) then
       return 0;
    else begin
            count = 0;
            p = C;
            repeat                    //不能用while迴圈!!∵一定至少要做一次!!!
               count++;
               p = p→link;
            until p = C;
         end;
end
```

∴ Time: O(n)

Concatenate

- For "Single Linked List"
- 假設連結A, B串列成為C串列:
  - Case 1: A = nil and B≠nil, 則C = B.
  - Case 2: A ≠ nil and B = nil, 則C = A.
  - Case 3: A ≠ nil and B ≠ nil, 則**C = A+B**.
  - Case 4: A = nil and B = nil, 則C = nil.

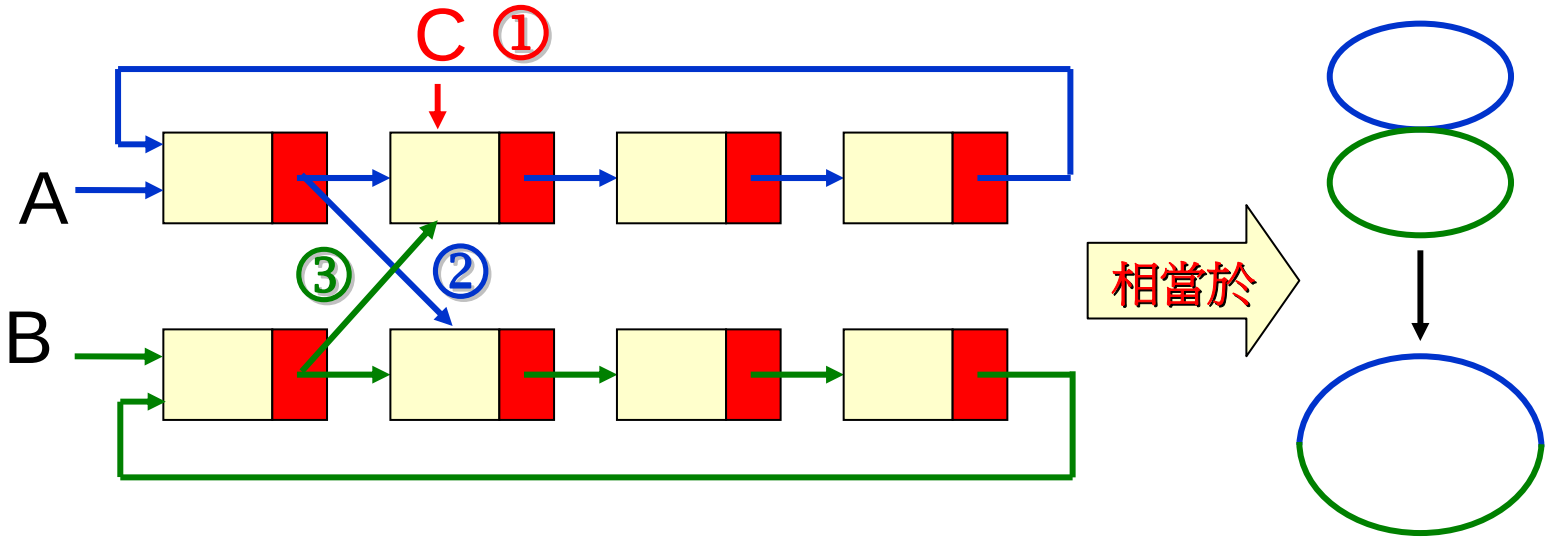


```
Procedure Con.two.S(A, B, C: pointer to S.L.)
begin
    C = nil;
    if (A = nil and B ≠ nil) then C = B;          //Case 1
    else if (A ≠ nil and B = nil) then C = A;     //Case 2
         else if (A ≠ nil and B ≠ nil) then       //Case 3
              begin
                   p = A;
                   ① while(p→link ≠ nil) do
                              p = p→link;
                   ② p→link = B;
                   ③ C = A;
              end;
    return C;     //回傳Case 1~4的結果
end
```

∴ Time: O(n)或O(m)

(m, n分別爲A, B串列的長度)
看p是在哪條串列上跑!!!

- For "Circular Linked List"
- 假設連結A, B串列成為C串列:
  - Case 1: A = nil and B≠nil, 則C = B.
  - Case 2: A ≠ nil and B = nil, 則C = A.
  - Case 3: A ≠ nil and B ≠ nil, 則**C = A+B**.
  - Case 4: A = nil and B = nil, 則C = nil.

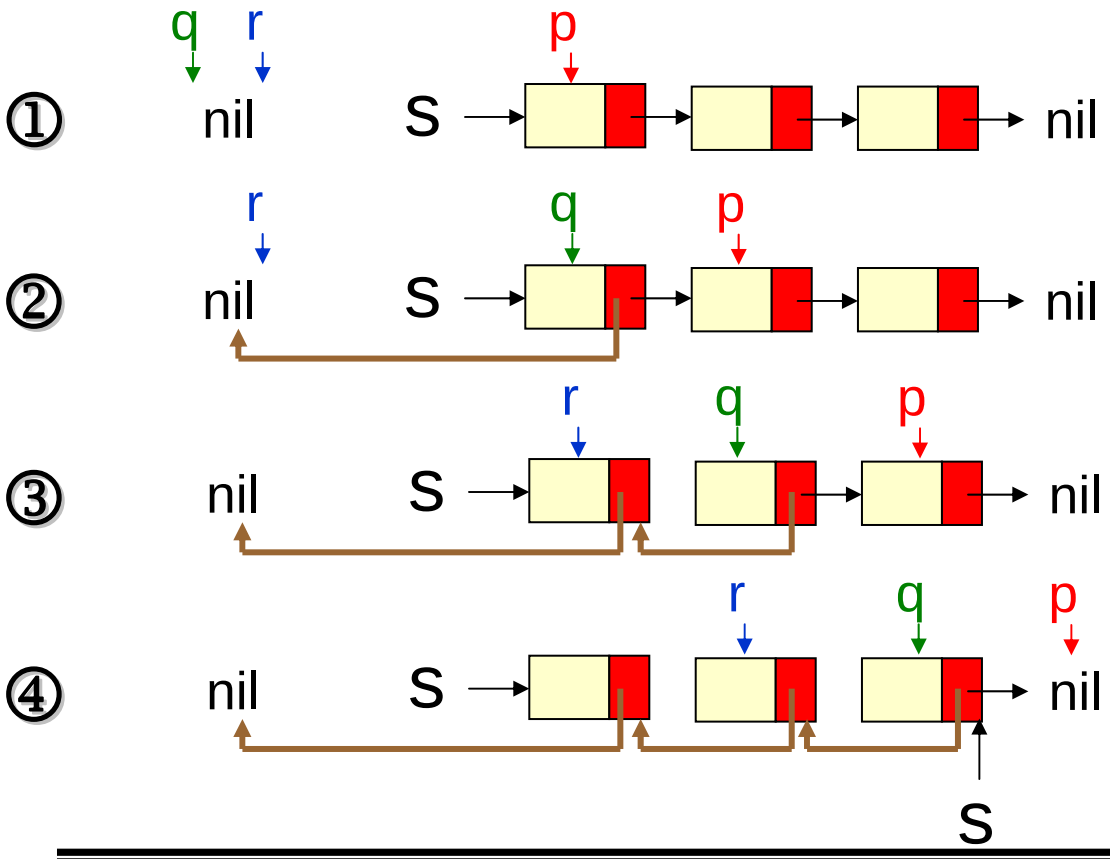


```
Procedure Con.two.S(A, B, C: pointer to C.L.)
begin
    C = nil;
    if (A = nil and B ≠ nil) then C = B;          //Case 1
    else if (A ≠ nil and B = nil) then C = A;     //Case 2
         else if (A ≠ nil and B ≠ nil) then       //Case 3
              begin
                   ① C = A→link;
                   ② A→link = B→link;
                   ③ B→link = C;
              end;
    return C;     //回傳Case 1~ 4的結果
end
```

∴ Time: O(1)

(∵沒有任何的迴圈指令，只有指標的改變)

- For "Single Linked List"
- 需要三個指標: r, q, p
  - r跟著q走，q跟著p走，p往前進

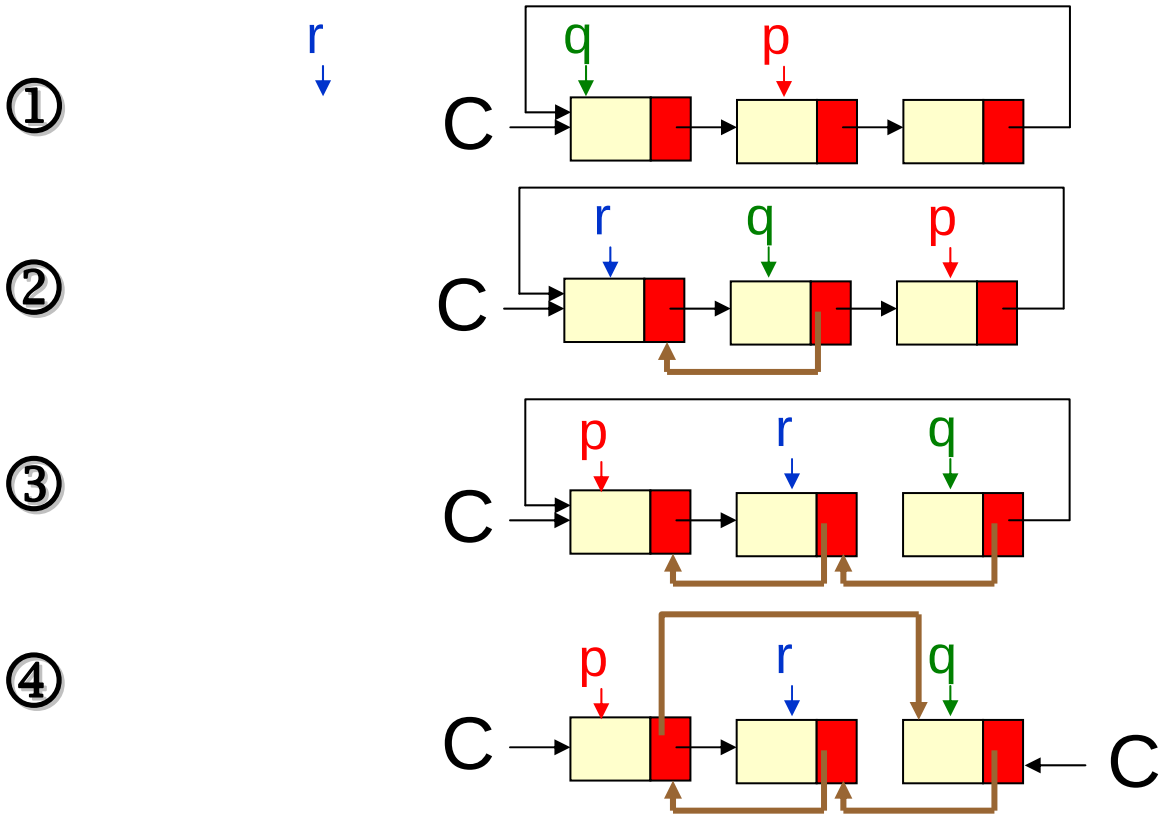


P跑到nil時結束，
同時S要指向q。

```
Procedure Invert S(S: pointer to S.L.)
begin
    p = S; q = nil;
    while (p ≠ nil) do
        begin
            r = q;    //r 一開始會在此被設為 nil
            q = p;
            p = p→link;
            q→link = r;
        end;
    S = q;    //S可能指向反轉後的第一個節點或是null list
end
```

∴ Time: O(n)

- For “Circular Linked List”
- 亦需要三個指標: r, q, p
  - r跟著q走，q跟著p走，p往前進



P跑到C時結束，
同時C要指向q。

```
Procedure Invert C(C: pointer to C.L.)
begin
    if (C ≠ nil) then
    begin
        p = C→link;     //p指向C的下一個node
        q = C;          //初值
        while (p ≠ c) do
        begin
            r = q;
            q = p;
            p = p→link;
            q→link = r;
         end;
         C→link = q;
         C = q;
    end;
end
```

∴ Time: O(n)

Array 與 Link List 的比較

	Link List	Array	
優			缺
缺			優

利用 Single link list 表示多項式

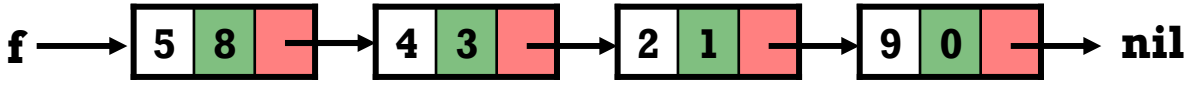
- 假設$f(x) = 5x^8+4x^3+2x+9$，請用Single link list 表示。

Ans:

- Node Structure 設計如右：

  - coe: Coefficient(係數); exp: Exponent(指數); link: Pointer

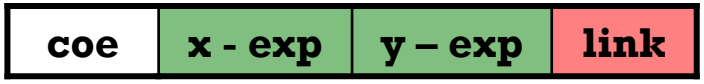
- 表示如下：



- 假設$f(x) = 5x^4y^2+8x^3y+9xy$，請用Single link list 表示。

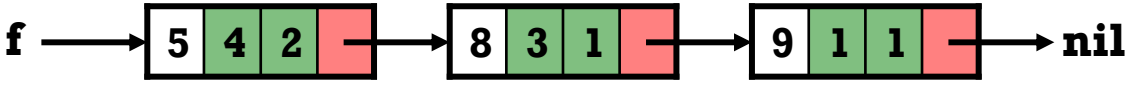
Ans:

- Node Structure 設計如右：



  - coe: Coefficient(係數); exp: Exponent(指數); link: Pointer

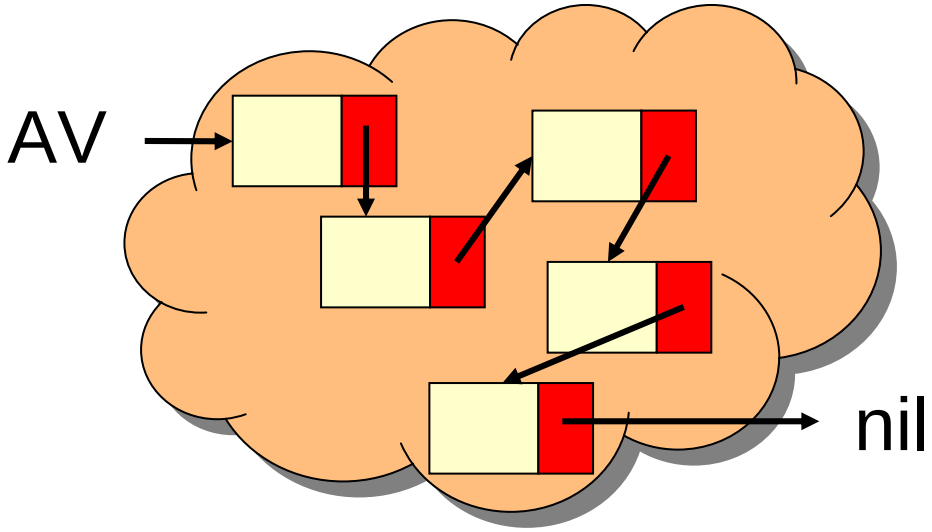
- 表示如下：



補　充

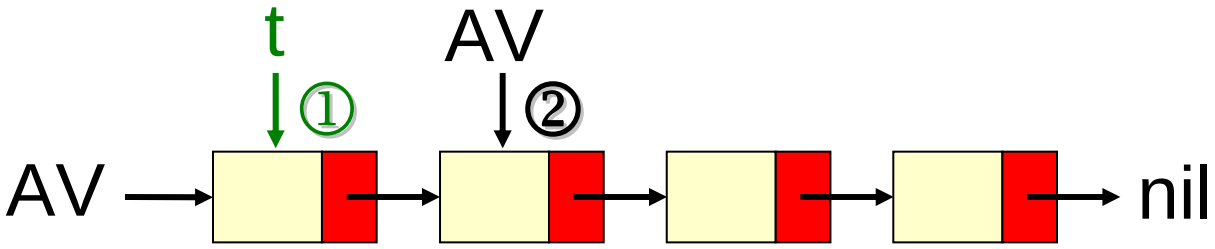
Storage Pool

- Def: **可用節點 (Free Node)** 之集合 (集中管理之處)，通常O.S.以**Single link list**來管理Free Nodes，稱為**AV-list** (Available list).



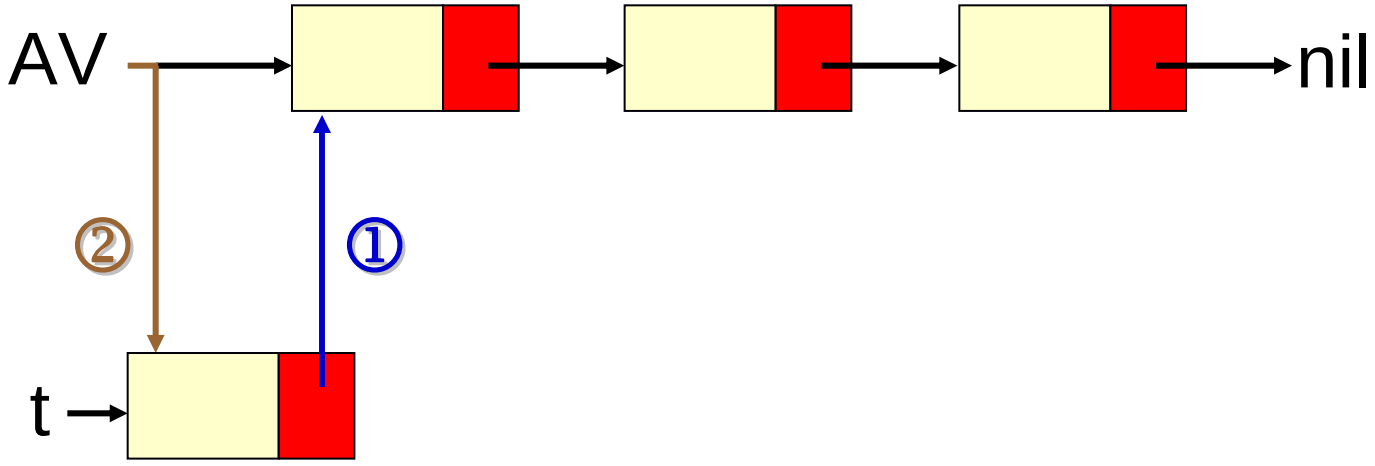
- 需提供二個運作:
  - **New(t)**: 相當於刪除AV-list中Free Node. (if AV-list ≠ nil)
  - **Ret(t)**: 相當於Insert節點到AV-list中.

New(t)




```
begin
    if (AV ≠ nil) then
      begin
          ① t = AV;
          ② AV = AV→link;
       end;
    else
       print("No Free Node.");
end
```

Ret(t)



begin

① **t→link = AV;**

② **AV = t;**

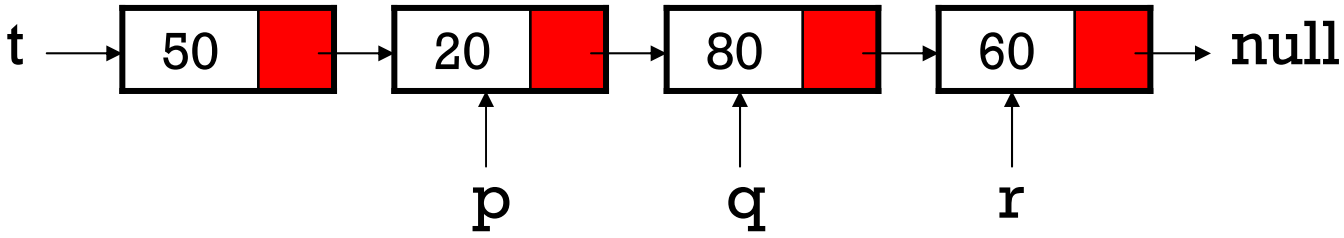
end

存取節點資料的表示方式

- 若單一節點是由指標t指向該節點:

  - t→Data:存取指標t所指向之節點中的data值
  - t → Link:存取指標t所指向之節點中的link值

- Ex:



- Print(p→Data) = __
- Print((p→Link)→Data) = __
- r相當於 _____ 或 _____
- 將p的下一個node改指向r節點 =_____